Xen Zynq Distribution

User's Manual - BETA

Xilinx-XenZynq-DOC-0001 v0.4 September 25, 2015

Revision History

The following table shows the revision history for this document.

Date MM/DD/YYYY	Version	Changes
03/30/2015	0.1	Converted Alpha Release Document using the Xilinx Template
04/22/2015	0.2	Updated steps and release to work with the beta version of petalinux
06/22/2015	0.3	Fixed the numbering scheme and added a section on non linux guests
09/24/2015	0.4	Added pass-through and alternative guest filesystem sections

Table of Contents

Chapter 1 Introduction 5
1.1. Introduction to the Xilinx Zynq UltraScale+ Multiprocessor System-on-Chip (MPSoC) Platform 5
1.2. Introduction to Xen Zynq Distribution 5

Chapter 2 Board Setup 6
2.1. Section To Be Written 6

Chapter 3 Host Setup - QEMU 7
3.1. QEMU Introduction 7
3.2. Release Image 7
3.3. Setting up host OS 7
3.3.1. Required Tools 8
3.3.2. Required Libraries 8
3.4. Installing the Image 8
3.4.1. Booting the emulated System 10
3.4.2. Booting a subdomain 13
3.4.3. Copying a guest 14
3.4.4. Booting guests with alternate file systems 17

Chapter 4 Building From Source 17
4.1. Environment Setup and Build Process 17
4.2. Build Dom0 Linux Kernel, Xen, U-Boot, & FSBL 17
4.3. Build the Dom0 File System 18
4.4. Build Device Tree Blob 20
4.5. Building and Installing the Guest 20
4.5.1. Build the Guest Domain Kernel 20
4.5.2. Build the Guest Domain File System Using BuildRoot 20
4.5.3. Install the BuildRoot Guest Image into the Dom0 File System 21
4.5.4. Alternate 1: Use Ubuntu Core FS 23
4.5.5. Alternate 2: Use Linaro OpenEmbedded FS 25
4.6 Converting dom0 File System Image 25
4.7 Running the System 26
4.8 Creating more guests 26

Chapter 5 Target Board 28

5.1. Section To Be Written 28

Chapter 6 Xen on Zynq 29
6.1. Xen Boot Process 29
6.2. xl – Interfacing to Xen 29
6.2.1. Listing Domains 29
6.2.2. Creating a guest domain 29
6.2.3. Shutting down or destroying a guest domain 30
6.2.4. Switching between domains 30

Chapter 7 A New Guest 31
7.1. Adding a new guest to the XZD 31
7.2. Starting the new guest 32

Chapter 8 Interacting with I/O Devices 33
8.1. Ethernet Pass-through 33
8.1.1. Introduction 33
8.1.2. Modifying the Xen Device Tree 33
8.1.3. Modifying the Domain Configuration File 36
8.1.4. Creating a Partial Device Tree 36
8.1.5. Communicating with the Domains 38

Chapter 9 Troubleshooting 39
9.1. Section To Be Written 39

Chapter 10 Additional Support Solutions 40
10.1. Current Support Options 40

1.1. Introduction to the Xilinx Zynq UltraScale+ Multiprocessor System-on-Chip (MPSoC) Platform

The Xilinx Zynq UltraScale+ Multiprocessor System-on-Chip (MPSoC) is a fully featured processing platform. Xilinx provides an overview of the Zynq UltraScale+ MPSoC here, zynq-ultrascale-mpsoc. The Zynq UltraScale+ MPSoC has many advanced features including a quad core ARM Cortex-A53, a dual core ARM Cortex–R5, the Mali-400MP2 GPU, and a highly configurable FPGA fabric. For more information on the ARM Cortex-A53, please visit cortex-a53-processor. The Zynq UltraScale+ MPSoC is capable of running the open source hypervisor Xen. Details on the Xen hypervisor are located at this web site, xenproject. These features make the new Zynq UltraScale+ MPSoC a strong choice for embedded applications, including aerospace & defense, medical, industrial, telecom, and many other application spaces.

1.2. Introduction to Xen Zynq Distribution

Xilinx Xen is the port of the Xen open source hypervisor, for the Xliinx Zynq UltraScale+ MPSoC. Xen is a type 1, or bare metal, hypervisor that can run on Intel and ARM platforms with hypervisor hardware extensions.

Xen provides the ability to run multiple operating systems on the same computing platform. Among the many uses of Xen, it allows the system designer to combine computing platforms into a single platform, conserving size, weight, and power (SWaP). Xilinx has ported the Xen hypervisor to run on the Zynq UltraScale+ MPSoC .

The Xen Zynq Distribution is released under the GNU General Public License version 2 (GPL2). You are free to use the Xen Zynq Distribution in any manner you see fit. You are free to make modifications and use them privately, without ever releasing them. If you decide to release any modifications to the public in some way, the license requires that you make the modified source code available to the program's users.

In addition to reducing SwaP, there are further advantages to using a hypervisor. Additional benefits are gained when using a hypervisor in applications that require a high level of safety and security. A hypervisor has the ability to isolate operating systems (OS) and provide the capability to run on a system where various OS have a different level of safety requirement or certification. Adding new applications in a new OS would theoretically only need to be certified as opposed to requiring the entire system to be certified again.

Security in a hypervisor can be designed in a similar fashion. Using a hypervisor, one can isolate various levels of security from each other, keeping confidential, secret, and top secret data from being accessible to unauthorized users virtual machines. Additionally, certification costs can be reduced as only the new application and OS needs to be certified since the hypervisor provides strict isolation between the OS.

Another benefit of a hypervisor that might not be immediately apparent is in designing systems where legacy applications and OS are not trusted and might lead to unexpected crashes. A bug causing a crash in an application or even in an OS will not interfere with the other applications and OS running in different virtual machines.

2.1. Section To Be Written

When the Zynq UltraScale+ MPSoC hardware is ready for release and shipped to customers, this chapter will contain instructions on how to setup the Zynq UltraScale+ MPSoC board.

3.1. QEMU Introduction

QEMU is used to emulate and virtualize hardware on a host computer. This allows a user to create and test software against the UltraScale+ MPSoC architecture. In many cases, access to actual hardware may be limited and using a virtualized system provides a significant cost savings while enabling the embedded engineer to develop applications specific for the hardware platform. QEMU is capable of emulating hardware ranging from x86 to PowerPC and, and in this case, ARM processors. One can download the QEMU source code and build for any number of supported hardware architectures.

The Xen Zynq Distribution environment consists of a version of QEMU built specifically for the Zynq UltraScale+ MPSoC processor. When the development system setup described in section 4 is started, a QEMU instance boots and is passed arguments that tell QEMU to emulate the quad core ARM Cortex-A53 hardware for the Zynq UltraScale+ MPSoC.

3.2. Release Image

The release image is a compressed TAR archive. The archive contains a prepackaged image that will allow the engineer to run the Xen hypervisor with a set of prepackaged domains. The figure below illustrates the contents of the beta release image with the included components needed to run the development system.

Once the archive is unpackaged, the directory structure will contain the following structure.

- *XenZynqDist-Beta_09_24_2015*
  - *dist*
    - *hw-description*
    - *images*
      - *linux*
    - *subsystems*
  - *XenZynqDist-Beta_09_24_2015.bsp*

3.3. Setting up host OS

These instructions are intended to be run on an x86_64 Ubuntu 12.04 host.

3.3.1. Required Tools

You will need git, several tools in Ubuntu's build-essential package, and others to complete this build process.

```
$ sudo apt-get install -y build-essential git mercurial dos2unix gawk tftpd-hpa flex bison unzip
```

3.3.2. Required Libraries

Install these additional libraries prior to PetaLinux installation.

Library	**Ubuntu Package Name**
ncurses terminal library	ncurses-dev
32-bit zlib compression library	lib32z1
32-bit GCC support library	lib32gcc1
32-bit ncurses	lib32ncurses5
32-bit Standard C++ library	lib32stdc++6
32-bit selinux library	libselinux1:i386

```
$ sudo apt-get install -y ncurses-dev lib32z1 lib32gcc1 lib32ncurses5 lib32stdc++6 libselinux1:i386
```

3.4. Installing the Image

There should be at least 10GB of free disk space available to decompress the archive and run the included scripts. The following setup was tested on an x84_64 PC running native Ubuntu 12.04 with 6 Gb DDR2 Memory, and a Core 2 Quad Q6600 processor.

If a developer wants to install/run the beta:

1. *Copy the release image to an appropriate location on the host system.*
2. *Open a bash terminal on the host system, and navigate to the image's location.*
3. *Extract the image into the directory with the following command:*

   ```
   $ tar -xvzf XenZynqDist-Beta_09_24_2015.tgz
   ```

4. *Create an envirnronment variable, $RELEASE_DIR*

   ```
   $ export RELEASE_DIR=`pwd`/XenZynqDist-Beta_09_24_2015
   ```

5. *One should also install the petalinux tools. The petalinux build environment requires that you link /bin/sh to /bin/bash.*

```
$ cd /bin
$ sudo rm sh
$ sudo ln -s bash sh
```

6. *Close, and reopen your terminal.*
7. *Reset the RELEASE_DIR variable*

```
$ export RELEASE_DIR=`pwd`/XenZynqDist-Beta_09_24_2015
```

8. *Download petalinux-v2015.SW_Beta2-final-installer.run from the Xilinx Early Access Lounge to the $RELEASE_DIR directory.*
9. *Once downloaded, the petalinux installer must be made executable.*

```
$ cd $RELEASE_DIR
$ chmod u+x petalinux-v2015.SW_Beta2-final-installer.run
```

10. *Install petalinux in your release directory by running the following command (This will require accepting a license agreement).*

```
$ ./petalinux-v2015.SW_Beta2-final-installer.run
```

11. *Once petalinux is installed, source the petalinux script using the following command.*

```
$ source petalinux-v2015.SW_Beta2-final/settings.sh
```

12. *Create tftpboot folder and install prebuilt binaries using the following commands*

```
$ sudo mkdir -p /tftpboot
$ sudo chmod 777 /tftpboot
$ cp $RELEASE_DIR/dist/images/linux/* /tftpboot/
```

3.4.1. Booting the emulated System

1. *Once the images have been installed correctly, they can be booted using QEMU. Boot QEMU using the following commands:*

```
$ cd $RELEASE_DIR/dist
$ petalinux-boot --qemu --u-boot --qemu-args "-drive
file=images/dom0.qcow,format=qcow2,id=sata-drive -device ide-drive,drive=sata-
drive,bus=ahci@0xFD0C0000.0 -net nic,vlan=0 -net user,vlan=0 -net nic,vlan=0"
```

   Running the command above will start the prepackaged images on the target machine. In this case the target is QEMU, running on the host computer. While the script is running, there will be kernel messages written to the stdout. They can be ignored.

   Executing the command will boot a QEMU system which emulates the Zynq UtraScale+ MPSoC, and necessary hardware. Once the emulated system is initialized, it will load the first stage boot loader (FSBL), which bootstraps the second stage boot loader, U-Boot, in QEMU's memory (RAM). The script will then load the dom0 file system as a virtual hard drive.

   At this point, QEMU will start booting the emulated Zynq UltraScale+ MPSoC. Once the FSBL has completed, U-Boot then takes over and loads the Xen kernel followed by the dom0 kernel. To accomplish this, U-Boot transfers the Xen kernel, the Xen device tree blob (DTB), and the dom0 kernel into QEMU's RAM. Once the images have been placed in memory and Xen has booted, boot up responsibilities transfer to Xen whose main priority is booting the dom0 kernel. The script passes in the dom0.img file to QEMU as a SATA device, and then QEMU uses that to emulate the hard drive.

   Once all of the above has been completed you will be presented with the dom0 login prompt.

2. *Log into the system using root/root as the username and password.*

   Once these steps have been completed, Xen and a virtual machine, specifically a Linux dom0 will be running on an emulated Xilinx UltraScale+ MPSoC.

   The image in Figure 2 visually shows the QEMU setup with Xen and the two domains included in the setup that we are walking through here.

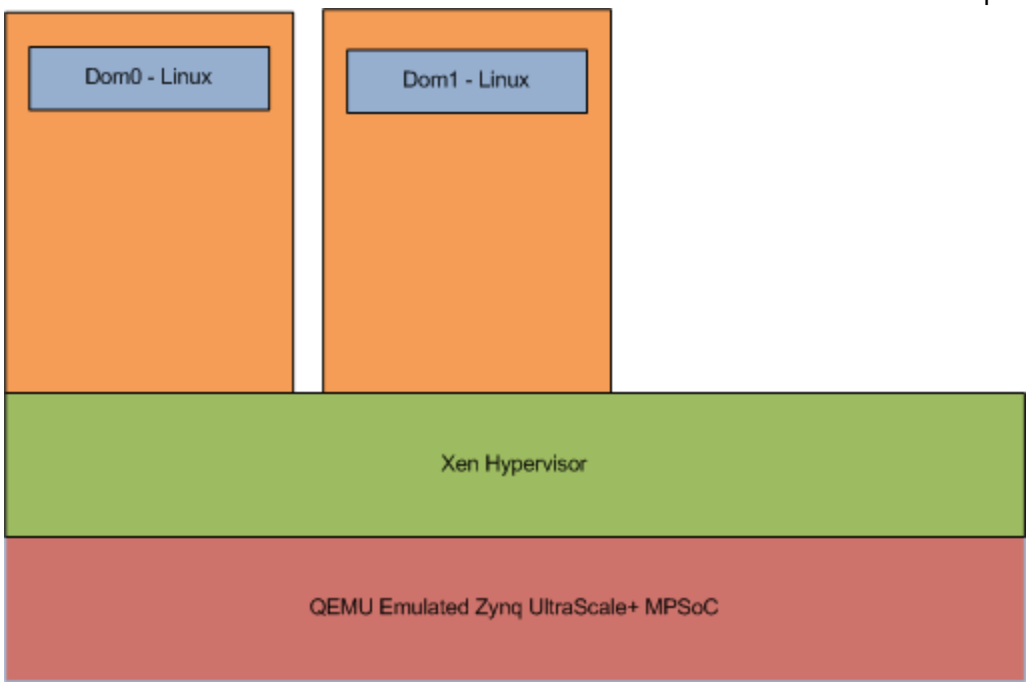

Figure 2. System Architecture

Now that QEMU has booted the emulated hardware, we need to make sure that Xen is running on the emulated hardware. To test that Xen is running and display diagnostic information, use the 'xl info' command. The expected output is shown on the following page.

```
[root@xilinx-buildroot ~]# xl info

host                   : xilinx-buildroot
release                : 3.18.0+
version                : #1 SMP Mon Mar 23 16:24:25 EDT 2015
machine                : aarch64
nr_cpus                : 1
max_cpu_id             : 127
nr_nodes               : 1
cores_per_socket       : 1
threads_per_core       : 1
cpu_mhz                : 50
hw_caps                :
00000000:00000000:00000000:00000000:00000000:00000000:00000000:00000000
virt_caps              :
total_memory           : 1024
free_memory            : 478
sharing_freed_memory   : 0
sharing_used_memory    : 0
outstanding_claims     : 0
free_cpus              : 0
xen_major              : 4
xen_minor              : 6
xen_extra              : .0-unstable
xen_version            : 4.6.0-unstable
xen_caps               : xen-3.0-aarch64 xen-3.0-armv7l
xen_scheduler          : credit
xen_pagesize           : 4096
platform_params        : virt_start=0x200000
xen_changeset          : Fri Jul 25 14:45:19 2014 +1000 git:b7c4d71
xen_commandline        : console=dtuart dtuart=serial0 dom0_mem=512M bootscrub=0
timer_slop=0
cc_compiler            : aarch64-buildroot-linux-gnu-gcc (Buildroot 2015.02-rc1-dirty) 4
cc_compile_by          : arlx
cc_compile_domain      :
cc_compile_date        : Mon Mar 23 16:03:02 EDT 2015
xend_config_format     : 4
```

If Xen is not properly installed or running, you will receive an error similar to:

```
xl: command not found
```

If you receive this error, then your terminal context is most likely not in the QEMU emulated environment. Make sure QEMU's terminal is in focus. If you continue to receive error, then you have booted a QEMU emulated system without Xen somehow. Please re-extract the release image, and try again.

To view the list of running domains, use the 'xl list' command

```
# xl list

Output:
Name                                        ID   Mem VCPUs      State   Time(s)
dom0                                         0   512     1     r-----     133.4
```

You are now free to test and experiment on the QEMU emulated system!

3.4.2. Booting a subdomain

The necessary components for your first subdomain (guest) have already been created. They are stored in dom0's file system:

```
Component                    Location in dom0's file system
Guest Linux Kernel           /root/Dom1-Kernel
Guest File System Image      /root/Dom1.img
Guest Domain Configuration   /etc/xen/dom1.cfg
```

Staying in the QEMU terminal, we will prepare to run a guest domain. The additional domain is already included in the archive we have been working with and will now prepare to run it on the Xen hypervisor. To do that type the following commands in dom0's console:

1. *Mount the guest's file system to a loop device in domain 0.*

   ```
   # losetup /dev/loop0 /root/Dom1.img
   ```

   You should receive no output if the command succeeds.

2. *Boot another domain (dom1), and connect to its console*

   ```
   # xl create -c /etc/xen/dom1.cfg
   ```

   The -c flag will automatically attach dom1's console. That is, once this command is executed, you will be logging into dom1. In order to return to dom0's console while leaving the guest (dom1) running, you can press CTRL-] to get back to dom0.

3. *Login using 'root' and 'root' for the username and password.*

   Since your guest is not a privileged domain, typing 'xl info' will output less detailed information, and 'xl list' will generate an error as it can only be run in dom0.

   ```
   # xl info

   Output:
   host                   : (none)
   release                : 3.18.0+
   version                : #1 SMP Tue Feb 24 13:00:31 EST 2015
   machine                : aarch64
   libxl: error: libxl.c:5044:libxl_get_physinfo: getting physinfo: Operation not permitted
   libxl_physinfo failed.
   libxl: error: libxl.c:5534:libxl_get_scheduler: getting domain info list: Operation not
   permitted
   xen_major              : 4
   xen_minor              : 6
   xen_extra              : .0-unstable
   xen_version            : 4.6.0-unstable
   xen_caps               : xen-3.0-aarch64 xen-3.0-armv7l
   xen_scheduler          : (null)
   xen_pagesize           : 4096
   platform_params        : virt_start=0x200000
   xen_changeset          : Tue Feb 24 22:48:17 2015 +1000 git:03e2f92
   xen_commandline        : console=dtuart dtuart=serial0 dom0_mem=512M bootscrub=0
   timer_slop=0
   cc_compiler            : aarch64-buildroot-linux-gnu-gcc (Buildroot 2015.02-rc1-dirty) 4
   cc_compile_by          : arlx
   ```

```
cc_compile_domain      :
cc_compile_date        : Fri Mar 13 15:46:51 EDT 2015
xend_config_format     : 4
```

```
# xl list

Output:
libxl: error: libxl.c:669:libxl_list_domain: getting domain info list: Operation not
permitted
libxl_list_domain failed.
```

The system is now running Xen and two domains or virtual machines: dom0 and dom1.

If you want to return to dom0's console while leaving the guest running, you may press CTRL-]. This will close the internal console connection, and bring dom0 back into focus within QEMU's terminal.

To reconnect to a guest terminal, use the "xl console" command

```
[root@xilinx-buildroot ~]# xl console dom1
```

It is important that you are aware which guest you are issuing commands to. Pay careful attention to the hostname listed in the terminal:

```
[root@xilinx-buildroot ~]#        dom0
[root@(none) ~]#                  dom1
```

3.4.3. Copying a guest

Both dom0 (control) and dom1 (guest) are included in the archive. You can easily boot a second guest domain, for a total of three domains including dom0, by making copies of dom1's components.

1. *Make sure that domain 1 is powered down before we copy its kernel and file system.*

```
[root@xilinx-buildroot ~]# xl console dom1
[root@(none) ~]# poweroff
```

The guest will shutdown, and the system should return to domain0's console automatically.

Make sure that the guest is completely shutdown by using the 'xl list' command. Dom1 should **NOT** have an entry. If you do see an entry for dom1, then this means that dom1 is still shutting down. Wait for approximately 15 or 20 seconds and try the command again. The results should appear similar to the below output.

```
[root@xilinx-buildroot ~]# xl list

Output:
Name                                        ID   Mem VCPUs State  Time(s)
dom0                                         0   512     1     r-----     259.7
```

2. *Copy the dom1 FS image.*

```
[root@xilinx-buildroot ~]# cp /root/Dom1.img /root/Dom2.img
```

This file is 1Gb, and will take a while to copy on the emulated SATA device.

3. *Copy the dom1 kernel.*

```
[root@xilinx-buildroot ~]# cp /root/Dom1-Kernel /root/Dom2-Kernel
```

4. *Copy the dom1 configuration file.*

```
[root@xilinx-buildroot ~]# cp /etc/xen/dom1.cfg /etc/xen/dom2.cfg
```

5. *Edit the new dom2 configuration file.*

   You will need to:
   a. Rename the guest to be named dom2.
   b. Configure the guest to boot domain 2's kernel.
   c. Change the targeted loop device to allow two domains to run simultaneously.

   You can accomplish this change using vi or sed expressions.

```
[root@xilinx-buildroot ~]# vi /etc/xen/dom2.cfg
```

   or

```
[root@xilinx-buildroot ~]# sed -i 's/om1/om2/' /etc/xen/dom2.cfg
[root@xilinx-buildroot ~]# sed -i 's/loop0/loop1/' /etc/xen/dom2.cfg
```

6. Verify that the changes have been made to the appropriate files.

```
[root@xilinx-buildroot ~]# cat /etc/xen/dom2.cfg

# =====================================================================
# Example PV Linux guest configuration
# =====================================================================

...

# Guest name
name = "dom2"

...

# Kernel image to boot
kernel = "/root/Dom2-Kernel"

...

# Disk Devices
# A list of `diskspec' entries as described in
# docs/misc/xl-disk-configuration.txt
disk = [ 'phy:/dev/loop1,xvda,w' ]
```

7. Mount the guest file systems to their respective loop devices in domain 0. The losetup command creates a device on which we can mount the file systems created.

```
# If you have already mounted /root/Dom1.img to /dev/loop0,
# there is no need to mount it again
[root@xilinx-buildroot ~]# losetup /dev/loop0 /root/Dom1.img
# The dom2 filesystem hasn't been mounted yet:
[root@xilinx-buildroot ~]# losetup /dev/loop1 /root/Dom2.img
```

8. *Start the domains (or virtual machines) with the following commands.*

```
[root@xilinx-buildroot ~]# xl create /etc/xen/dom1.cfg
```

When you leave the -c flag off of the domain creation command, you will receive the guests initial boot messages to dom0's standard out, while also keeping dom0's console in focus.

This chatter does not affect your standard input however it does make it a bit hard to type the next command. You might want to wait before issuing the next command. The last message should look similar to the below output (the '3' in the vif3.0 output is variable).

```
...
xenbr0: port 2(vif3.0) entered forwarding state
xenbr0: port 2(vif3.0) entered forwarding state
xenbr0: port 2(vif3.0) entered forwarding state
```

Hit enter to bring up a new line in the console, indicating your hostname. Make sure you are still in dom0's console, and that you didn't attach to the guest.

```
[root@xilinx-buildroot ~]#
[root@xilinx-buildroot ~]# xl create /etc/xen/dom2.cfg
```

You should see similar console chatter from dom2 booting as it reports to standard out.

The 'xl list' command will now show all three domains running. The ID values are sequential and will increase each time a domain is created. The ID numbers here might look different in your output.

```
[root@xilinx-buildroot ~]# xl list

Name                                        ID   Mem VCPUs      State   Time(s)
dom0                                         0   512     1     r-----     378.0
dom1                                         1   128     1     -b----      67.8
dom2                                         2   128     1     -b----      46.5
```

Guest domains should be shutdown carefully, as their file systems are easily corrupted if they are reset improperly or shutdown in the middle of an IO operation. There are two methods to shutdown a guest:

From domain0, you can request the Xen Kernel to send a shutdown signal to a guest:

```
[root@xilinx-buildroot ~]# xl shutdown dom1
```

You could also attach to dom1's console by executing the command 'xl console dom1' and send the poweroff command:

```
# To attach to dom1's console
[root@xilinx-buildroot ~]# xl console dom1
# While in dom1
[root@(none) ~]# poweroff
```

The poweroff command will function as expected within dom0's console as well, but you should make sure that all domains are properly shutdown before doing so.

```
[root@xilinx-buildroot ~]# poweroff
```

Once all the domains have been shutdown, you may terminate the instance of QEMU using the keyboard shortcut CTRL-A X. (Control-A, release, then X)

View other QEMU shortcuts by typing CTRL-A H (Control-A, release, then H, while QEMU is running).

3.4.4. Booting guests with alternate file systems

Two alternate guest filesystems and matching xen configuration files have also been provided, these are the Ubuntu Core file system and the Linaro flavored OpenEmbedded file system. Their images can be mounted and the guests booted using commands similar to those found in 3.4.2:

For the Ubuntu Core FS:

1. *Mount the guest's file system to a loop device in domain 0.*

```
# losetup /dev/loop1 /root/ubuntu-core-fs.img
```

2. *Boot another domain, and connect to its console*

```
# xl create -c /etc/xen/ubuntu-core-fs.cfg
```

For the Linaro OpenEmbedded FS:

1. *Mount the guest's file system to a loop device in domain 0.*

```
# losetup /dev/loop2 /root/linaro-openembedded-fs.img
```

2. *Boot another domain, and connect to its console*

```
# xl create -c /etc/xen/linaro-openembedded-fs.cfg
```

You can perform all of the same operations with these guests as described in 3.4.2 and 3.4.3

Chapter 4 Building From Source

4.1. Environment Setup and Build Process

If you have not executed the steps in section 3.4, in a terminal, set RELEASE_DIR to the directory path where the archive was decompressed. This variable will be used in several instructions so that you may copy-paste them.

```
$ export RELEASE_DIR=`pwd`/XenZynqDist-Beta_09_24_2015
```

4.2. Build Dom0 Linux Kernel, Xen, U-Boot, & FSBL

The following instructions assume that you are using the provided petalinux and qemu binaries.

TIP: *If you have not extracted the release, please visit chapter 3 - section 3 and follow the steps to decompress the archive into a directory. If you have not installed the petalinx tools, please refer to section 3.3.*

1. Once petalinux is installed, source the petalinux script using the following command.

```
$ cd $RELEASE_DIR
$ source petalinux-v2015.SW_Beta2-final/settings.sh
```

2. Create a project directory named 'XenZynqDist' using the BSP provided:

```
$ petalinux-create -t project -s XenZynqDist-Beta_09_24_2015.bsp -n XenZynqDist
```

   This project directory should have everything you need to build the Xen Zynq distribution.

3. Enter the project directory

```
$ cd $RELEASE_DIR/XenZynqDist
```

4. (Optional) If there is a need to configure the dom0 kernel, you can run the command below and select the features that need to be added.

   Added kernel configurations are beyond the scope of this document and should only be done if there is a specific reason.

   The default configuration will be sufficient to run as the dom0 kernel for this exercise.

```
# (This command is optional)
$ petalinux-config -c kernel
```

5. Use petalinux to build the linux kernel, xen, U-Boot, and FSBL images.

```
$ petalinux-build
```

TIP: *If you want more information during the build, you can use the verbose flag '–v' with the petalinux-build command.*

6. View your images in the 'images/linux/' directory

```
$ ls images/linux/
bl31.bin  Image      image.ub    System.map.linux  u-boot.elf    u-boot-s.elf  u-boot-
s.srec  xen.ub
bl31.elf  image.elf  system.dtb  u-boot.bin        u-boot-s.bin  u-boot.srec   vmlinux
zynqmp_fsbl.elf
```

4.3. Build the Dom0 File System

These instructions build the file system (FS) for the system domain, Dom0.

7. Clone the buildroot source and create the default configuration file for the Dom0 FS.

```
$ cd $RELEASE_DIR
$ git clone https://github.com/dornerworks/buildroot.git -b xen-guest-fs
$ cd buildroot
$ make zynq_ultra_mpsoc_dom0_defconfig
```

8. Run the menuconfig tool if there is specific functionality that needs to be built into the FS.

```
# (This command is optional)
$ make menuconfig
```

9. Once your configuration is complete, build the FS.

```
$ make
```

The make may run for up to one hour depending on the host system specifications.

The make command generates a FS tarball called rootfs.tar in the following location:

```
$ ls $RELEASE_DIR/buildroot/output/images
rootfs.cpio  rootfs.cpio.gz  rootfs.tar
```

10. Copy package configuration within buildroot

```
$ cp $RELEASE_DIR/buildroot/output/host/usr/bin/pkg-config
$RELEASE_DIR/buildroot/output/host/usr/bin/aarch64-buildroot-linux-gnu-pkg-config
```

11. Add buildroot compiler to your path

```
$ export PATH=$PATH:$RELEASE_DIR/buildroot/output/host/usr/bin/
```

12. Configure and build the Xen tools

```
$ cd $RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src
$ ./configure --host=aarch64-buildroot-linux-gnu --enable-tools
$ make dist-tools CROSS_COMPILE=aarch64-buildroot-linux-gnu- XEN_TARGET_ARCH=arm64
CONFIG_EARLY_PRINTK=ronaldo
```

13. Add the Xen tools to buildroot

```
$ cd $RELEASE_DIR
$ cp $RELEASE_DIR/buildroot/output/images/rootfs.tar dom0.rootfs.tar
$ fakeroot tar -rvf dom0.rootfs.tar --transform='s,usr/lib64,usr/lib,S' --
transform='s,var/log,tmp,S' --show-transformed-names  -
C$RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src/dist/install ./etc/ -
C$RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src/dist/install ./usr/ -
C$RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src/dist/install ./var/
```

14. Create and partition the dom0 image file:

```
$ dd if=/dev/zero of=$RELEASE_DIR/XenZynqDist/images/dom0.img bs=1G count=8
$ echo -e "n\np\n1\n\n\nt\n83\nw\n" | fdisk $RELEASE_DIR/XenZynqDist/images/dom0.img
```

15. Mount and format the Dom0 image file:

```
$ sudo losetup -o 1048576 /dev/loop0 $RELEASE_DIR/XenZynqDist/images/dom0.img
$ sudo mkfs.ext2 /dev/loop0
$ mkdir -p $RELEASE_DIR/tmp/dom0_fs
$ sudo mount /dev/loop0 $RELEASE_DIR/tmp/dom0_fs
```

16. Install the Dom0 FS onto the image file:

```
$ sudo tar -xvf dom0.rootfs.tar -C $RELEASE_DIR/tmp/dom0_fs
```

17. Unmount the Dom0 image file:

```
$ sudo umount /dev/loop0
$ sudo losetup -d /dev/loop0
```

4.4. Build Device Tree Blob

1. Create the device tree blob (DTB)

```
$ dtc -I dts -O dtb -o /tftpboot/xen.dtb $RELEASE_DIR/xen.dts
```

4.5. Building and Installing the Guest

4.5.1. Build the Guest Domain Kernel

If the guest kernel does not need to be configured differently from the dom0 kernel, then one can skip to step 5 in this section.

1. Change directory to the Xen Zynq Distribution

```
$ cd $RELEASE_DIR/XenZynqDist
```

2. Configure the guest kernel

```
$ petalinux-config -c kernel
```

3. Build a new kernel

```
$ petalinux-build -c kernel
```

4. Assemble the kernel image

```
# Assemble the image:
$ petalinux-build -x package
```

5. Copy the guest linux kernel image

```
$ cp $RELEASE_DIR/XenZynqDist/images/linux/Image $RELEASE_DIR/Dom1-Kernel
```

4.5.2. Build the Guest Domain File System Using BuildRoot

1. Change to the $RELEASE_DIR/buildroot directory. Create the default configuration file for a guest FS.

```
$ cd $RELEASE_DIR/buildroot
$ make zynq_ultra_mpsoc_guest_defconfig
```

2. Run the menuconfig tool if there is specific functionality that needs to be built into the FS.

```
$ make menuconfig
```

3. Once your configuration is complete, build the FS.

```
$ make
```

The make command generates a FS tarball called rootfs.tar in the following location:

```
$ ls $RELEASE_DIR/buildroot/output/images
rootfs.cpio  rootfs.cpio.gz  rootfs.tar
```

4. If the Xen tools are not built, then follow the steps to build the Xen-Tools in the step *Configure and build the Xen tools* to create the tools. Following this, then execute the command below.

```
$ cd $RELEASE_DIR
$ cp $RELEASE_DIR/buildroot/output/images/rootfs.tar dom1.rootfs.tar
```

```
$ fakeroot tar -rvf dom1.rootfs.tar --transform='s,usr/lib64,usr/lib,S' --
transform='s,var/log,tmp,S' --show-transformed-names  -
C$RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src/dist/install ./etc/ -
C$RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src/dist/install ./usr/ -
C$RELEASE_DIR/XenZynqDist/components/apps/xen/xen-src/dist/install ./var/
```

4.5.3. Install the BuildRoot Guest Image into the Dom0 File System

To construct the configuration files for a new domain and then insert those files into the dom0 File system, follow the steps below:

1. Attach dom0's FS to a loop device, and mount it to a temporary directory.

```
$ cd $RELEASE_DIR
$ sudo losetup -o 1048576 /dev/loop0 $RELEASE_DIR/XenZynqDist/images/dom0.img
$ mkdir -p $RELEASE_DIR/tmp/dom0_fs
$ sudo mount /dev/loop0 $RELEASE_DIR/tmp/dom0_fs
```

2. Move the Dom1-Kernel into dom0's filesystem.

```
$ sudo mv $RELEASE_DIR/Dom1-Kernel $RELEASE_DIR/tmp/dom0_fs/root/Dom1-Kernel
```

3. Create a configuration file for your DomU. Below is the configuration file for Dom1. It contains additional options that will not be used in the rest of the demonstration.

   Move to the following directory and insert the following file in your desired manner:

```
$ cd $RELEASE_DIR/tmp/dom0_fs/etc/xen/
```

```
# =====================================================================
# Example PV Linux guest configuration
# =====================================================================
#
# This is a fairly minimal example of what is required for a
# Paravirtualised Linux guest. For a more complete guide see xl.cfg(5)

# Guest name
name = "dom1"

# 128-bit UUID for the domain as a hexadecimal number.
# Use "uuidgen" to generate one if required.
# The default behavior is to generate a new UUID each time the guest is started.
#uuid = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"

# Kernel image to boot
kernel = "/root/Dom1-Kernel"

# Kernel command line options
extra = "console=hvc0 earlyprintk=xenboot root=/dev/xvda1 rw"

# Initial memory allocation (MB)
memory = 128

# Maximum memory (MB)
```

```
# If this is greater than `memory' then the slack will start ballooned
# (this assumes guest kernel support for ballooning)
#maxmem = 512

# Number of VCPUS
vcpus = 1

# Network devices
# A list of 'vifspec' entries as described in
# docs/misc/xl-network-configuration.markdown
vif = [ 'bridge=xenbr0' ]

# Disk Devices
# A list of `diskspec' entries as described in
# docs/misc/xl-disk-configuration.txt
disk = [ 'phy:/dev/loop0,xvda,w' ]
```

***TIP: Each** domain requires a configuration file that Xen can use to boot the domain. The configuration files are generally located in /etc/xen, however they can be located anywhere one chooses.*

4. Create a blank 1Gb file to be used as the Guest's (Dom1's) FS Image

   Take note of the Output File name (of=<Name>) you are choosing in the following command:

```
$ sudo dd if=/dev/zero of=$RELEASE_DIR/tmp/dom0_fs/root/Dom1.img bs=1M count=1024
$ cd $RELEASE_DIR
```

5. Create a partition table on the image, and use the entire image file as a Linux partition

```
$ sudo sh -c 'echo -e "n\np\n1\n\n\nt\n83\nw\n" | fdisk tmp/dom0_fs/root/Dom1.img'
```

6. Make a temporary mounting directory for the Guest FS image.

```
$ mkdir -p $RELEASE_DIR/tmp/dom1_fs
```

7. Attach the Guest FS image to a loop device.

```
$ sudo losetup -o 1048576 /dev/loop1 $RELEASE_DIR/tmp/dom0_fs/root/Dom1.img
```

8. Format the Guest FS image using 'mkfs.ext2'

```
$ sudo mkfs.ext2 /dev/loop1
```

9. Mount the Guest FS image to the temporary mounting directory.

```
$ sudo mount /dev/loop1 $RELEASE_DIR/tmp/dom1_fs
```

10. Un-pack the files for the FS.

```
$ sudo tar -xvf dom1.rootfs.tar -C $RELEASE_DIR/tmp/dom1_fs
```

11. Unmount and detach the Guest and dom0 filesystems.

```
$ sudo umount /dev/loop1
$ sudo losetup -d /dev/loop1

$ sudo umount /dev/loop0
$ sudo losetup -d /dev/loop0
```

CAUTION! *Due to how petalinux currently works, anytime a new guest domain is built that is configured differently from the dom0, a new dom0 must be built as the original one will be over-written*

4.5.4. Alternate 1: Use Ubuntu Core FS

The following are instructions for using the Ubuntu Core file system as an *alternative* to the steps described in 4.5.2 and 4.5.3 for the BuildRoot file system. If you have completed 4.5.2 and 4.5.3 or 4.5.5 please continue to 4.6 to finish the process of booting the guest.

1. Obtain the file system contents from the following link:

   http://cdimage.ubuntu.com/ubuntu-core/releases/14.04/release/ubuntu-core-14.04.3-core-arm64.tar.gz

2. Attach dom0's FS to a loop device, and mount it to a temporary directory.

```
$ cd $RELEASE_DIR
$ sudo losetup -o 1048576 /dev/loop0 $RELEASE_DIR/XenZynqDist/images/dom0.img
$ mkdir -p $RELEASE_DIR/tmp/dom0_fs
$ sudo mount /dev/loop0 $RELEASE_DIR/tmp/dom0_fs
```

3. Use the following commands to create a file system image for holding the Ubuntu Core contents: (For more information see Section 4.5.3 steps 4-10)

```
$ sudo dd if=/dev/zero of=$RELEASE_DIR/tmp/dom0_fs/root/ubuntu-core-fs.img bs=1M count=1024
$ cd $RELEASE_DIR
$ sudo sh -c 'echo -e "n\np\n1\n\n\nt\n83\nw\n" | fdisk tmp/dom0_fs/root/ubuntu-core-
fs.img'
$ mkdir -p $RELEASE_DIR/tmp/ubuntu-core-fs
$ sudo losetup /dev/loop1 $RELEASE_DIR/tmp/dom0_fs/root/ubuntu-core-fs.img
$ sudo mkfs.ext2 /dev/loop1
$ sudo mount /dev/loop1 $RELEASE_DIR/tmp/ubuntu-core-fs
```

4. Untar the Ubuntu Core FS files into the into the newly created image file in the dom0 FS

```
$ tar -xvpzf $DOWNLOAD_LOCATION/ubuntu-core-14.04.3-core-arm64.tar.gz -C $RELEASE_DIR/tmp
/ubuntu-core-fs
```

5. Configure the Ubuntu Core image to work with Xen

   First, using a text editor, add the file `$RELEASE_DIR/tmp/ubuntu-core-fs/etc/init/hvc0.conf` with the following contents to enable login:

```
# hvc0 - getty
#
# This service maintains a getty on tty1 from the point the system is
# started until it is shut down again.

start on stopped rc RUNLEVEL=[2345] and (
            not-container or
            container CONTAINER=lxc or
            container CONTAINER=lxc-libvirt)

stop on runlevel [!2345]
```

```
respawn
exec /sbin/getty -8 38400 hvc0
```

Second, edit the root entry in the $RELEASE_DIR/tmp/ubuntu-core-fs/etc/passwd file to look as follows. This will set the root user password to empty and allow you to login in as the root user:

```
root::0:0:root:/root:/bin/bash
```

6. Add a configuration file for your domU as detailed in 4.5.3 step 3, name the file ubuntu-core.cfg and make the following edits:
   - Set the domU to have a different name:

```
name = “ubuntu-core”
```

   - Set the disk line to read:

```
disk = [ 'phy:/dev/loop1,xvda,w' ]
```

7. Unmount and detach the Ubuntu Core and dom0 filesystems.

```
$ sudo umount /dev/loop1
$ sudo losetup -d /dev/loop1

$ sudo umount /dev/loop0
$ sudo losetup -d /dev/loop0
```

Usage Note: On first boot of the Ubuntu Core FS guest, log in with user name “root” with no password. Once logged in, set the root password using the passwd command, it should look something like this:

```
Ubuntu 14.04.1 LTS localhost.localdomain hvc0

localhost login: root
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.18.0 aarch64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@localhost:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

4.5.5. Alternate 2: Use Linaro OpenEmbedded FS

The following are instructions for using the Linaro OpenEmbedded file system as an *alternative* to the steps described in 4.5.2 and 4.5.3 for the BuildRoot file system. If you have completed 4.5.2 and 4.5.3 or 4.5.4 please continue to 4.6 to finish the process of booting the guest.

1. Obtain the image from the following link:

   https://releases.linaro.org/15.06/openembedded/juno-lsk/lt-vexpress64-openembedded_minimal-armv8-gcc-4.9_20150620-722.img.gz

2. Attach dom0's FS to a loop device, and mount it to a temporary directory.

```
$ cd $RELEASE_DIR
$ sudo losetup -o 1048576 /dev/loop0 $RELEASE_DIR/XenZynqDist/images/dom0.img
$ mkdir -p $RELEASE_DIR/tmp/dom0_fs
$ sudo mount /dev/loop0 $RELEASE_DIR/tmp/dom0_fs
```

3. Untar the new FS image to the dom0 FS

```
$ tar -xvpzf lt-vexpress64-openembedded_minimal-armv8-gcc-4.9_20150620-722.img.gz -C
$RELEASE_DIR/tmp/dom0_fs/root/
```

   - For clarity and ease of use, rename the untared image file:

```
$ mv $RELEASE_DIR/tmp/dom0_fs/root/lt-vexpress64-openembedded_minimal-armv8-gcc-
4.9_20150620-722.img $RELEASE_DIR/tmp/dom0_fs/root/linaro-openembedded-fs.img
```

4. Add a configuration file for your domU as detailed in 4.5.3 step 3, name the file `linaro-openembedded.cfg` and make the following edits:

   - Set the domU to have a different name:

```
    name = "linaro-openembedded"
```

   - Set the "disk" line to read *(take note of xvdb!)*:

```
    disk = [ 'phy:/dev/loop2,xvdb,w' ]
```

   - Set the "extra" line to read as follows (*take note of xvdb2)*:

```
   /home/arlx/xzd/fs/dom0/etc/xen/linaro-openembedded-fs.cfg
```

5. Unmount and detach the dom0 filesystem.

```
$ sudo umount /dev/loop0
$ sudo losetup -d /dev/loop0
```

Usage Note: On first boot of the Linaro OpenEmbedded FS, use 'root' for both the username and password

4.6 Converting dom0 File System Image

The Dom0 FS image that was created is fairly large as a raw image file. To deal with that, the image is converted to a qcow2 image, which is a compressed image format that QEMU understands. To convert to this format, use the following command:

```
$ qemu-img convert -O qcow2 $RELEASE_DIR/XenZynqDist/images/dom0.img
```

```
$ RELEASE_DIR/XenZynqDist/images/dom0.qcow
$ rm $RELEASE_DIR/XenZynqDist/images/dom0.img
```

4.7 Running the System

1. Boot QEMU by running the following command

```
$ cd $RELEASE_DIR/XenZynqDist
$ petalinux-boot --qemu --u-boot --qemu-args "-drive
file=images/dom0.qcow,format=qcow2,id=sata-drive -device ide-drive,drive=sata-
drive,bus=ahci@0xFD0C0000.0"
```

2. Start your customized domain.

   For the standard file system created using BuildRoot:

```
[root@xilinx-buildroot ~]# losetup /dev/loop0 /root/Dom1.img
[root@xilinx-buildroot ~]# xl create /etc/xen/dom1.cfg –c
```

   If you're using the Ubunt Core FS or Linaro OpenEmbedded FS they can be started similarly:

```
[root@xilinx-buildroot ~]# losetup /dev/loop1 /root/ubuntu-core-fs.img
[root@xilinx-buildroot ~]# xl create /etc/xen/ubuntu-core.cfg –c
```

   And

```
[root@xilinx-buildroot ~]# losetup /dev/loop2 /root/linaro-openembedded.img
[root@xilinx-buildroot ~]# xl create /etc/xen/linaro-openembedded.cfg –c
```

TIP: *There are times when the domain is created in a paused state. To correct this problem, enter the following commands below:*

```
[root@xilinx-buildroot ~]# xl unpause dom1
[root@xilinx-buildroot ~]# xl console dom1
```

3. Verify that the new domain is running.

```
[root@xilinx-buildroot ~]# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
dom0                                         0   512     1     r-----      36.7
dom1                                         1   128     1     -b----      10.3
```

4.8 Creating more guests

To create more guests from source, just follow the steps in section 4.5, but anywhere "Dom1" is used, use the name of the new domain, such as Dom2. When the Dom0 FS image needs to be mounted, make sure to convert the qcow2 formatted image back to a raw formatted image with the following command

```
$ qemu-img convert -O raw $RELEASE_DIR/XenZynqDist/images/dom0.qcow
$RELEASE_DIR/XenZynqDist/images/dom0.img
```

Chapter 5 Target Board

5.1. Section To Be Written

This chapter will discuss how to build for various targets including QEMU and the Zynq UltraScale+ MPSoC. This section will be updated when more details on the hardware become available.

6.1. Xen Boot Process

Running Xen requires booting two kernels: the Xen kernel and the dom0 (or control) kernel, which at this point is a Linux kernel. Both kernels are loaded into the proper memory locations by the boot loader. Once the boot loader has finished the initialization of the system, it passes control of the boot process over to Xen.

Xen then performs some additional hardware initialization such as initializing the Zynq hardware so that Xen can map and handle requests from the device drivers used by the dom0 kernel. More details on the Xen boot process can be found on the Xen Wiki (http://www.xenproject.org/help/wiki.html).

Once Xen performs its initialization, it then loads the dom0 (usually Linux) kernel and the RAM disk into memory. The dom0 kernel is then booted by Xen inside the privileged virtual machine domain; from the perspective of the kernel and the user this boot process is identical to Linux booting directly on the system hardware. Once dom0 has booted, access to Xen can be configured for each unprivileged domain via the dom0 interface to Xen. Dom0 has special privileges allowing it to perform this configuration, among them being the ability to access the system hardware directly. It also runs the Xen management toolstack, briefly described below.

6.2. xl – Interfacing to Xen

xl provides an interface to interact with Xen. It is the standard Xen project supported toolstack provided with the Xen hypervisor for virtual machine configuration, management, and debugging.

6.2.1. Listing Domains

'xl list' is used to list the running domains and their states on the system.

```
# xl list

Output:
Name                        ID   Mem VCPUs      State  Time(s)
Domain-0                     0  2048     2     r-----    32.0
dom1                         1  1024     2     r-----     7.3
```

6.2.2. Creating a guest domain

The following command will start a new domain using the provided configuration file argument. The name of the domain is determined by the value set in the configuration file.

```
# xl create -c /etc/xen/dom1.cfg
```

6.2.3. Shutting down or destroying a guest domain

'xl shutdown' is the command that should be used to shut down a running guest domain on the system. It performs a graceful shutdown of the domain using the built-in shutdown features of the domain's operating system, allowing the domains files system to close safely, and releasing any hardware resources (RAM, CPU Time, etc.) back to the system.

```
# xl shutdown dom1
```

'xl destroy' Should only be used as a last resort on unresponsive domains that are not removed when given the 'xl shutdown' command. The destroy command does not perform a graceful shutdown and potentially could corrupt the guest domain's file system as well as any I/O devices to which the domain has been given access.

```
# xl destroy dom1
```

6.2.4. Switching between domains

To access the command-line interface of a running domain, use the following command:

```
# xl console <domain-name>
```

where <domain-name> is the name of the specific domain of interest (the names of all running domains can be viewed using the `xl list` command detailed above).

To return to the dom0 command-line interface, press the key combination: CTRL-].

7.1. Adding a new guest to the XZD

There are more than just Linux OS's available as guests for the Xen Zynq Distribution. This section will go over how to install and run one of these other guests. We will be using the XZD bare metal guest as an example. These instructions assume you have followed 3.4.4 already.

First, follow the *Xen Zynq Distribution (XZD) Bare Metal Guest How To Guide* to successfully build the XZD bare metal guest image, xzd_bare.img. Once you have your built image, it needs to be placed in the dom0 filesystem. To do that, perform the following:

1. Source the Petalinux toolset

```
$ cd $RELEASE_DIR
$ source petalinux-v2015.SW_Beta2-final/settings.sh
```

2. Convert the dom0 FS QCOW2 image to a RAW format

```
$ qemu-img convert -O raw $RELEASE_DIR/XenZynqDist/images/dom0.qcow
$RELEASE_DIR/XenZynqDist/images/dom0.img
```

3. Attach dom0's FS to a loop device, and mount it to a temporary directory.

```
$ sudo losetup -o 1048576 /dev/loop0 $RELEASE_DIR/XenZynqDist/images/dom0.img
$ mkdir -p $RELEASE_DIR/tmp/dom0_fs
$ sudo mount /dev/loop0 $RELEASE_DIR/tmp/dom0_fs
```

4. Move the bare metal image into dom0's filesystem.

```
$ sudo mv $RELEASE_DIR/xzd_bare.img $RELEASE_DIR/tmp/dom0_fs/root/xzd_bare1.img
```

5. Create a configuration file for your bare metal guest (/etc/xen/bare1.cfg). Below is an example configuration file.

   Move to the following directory and insert the following file in your desired manner:

```
$ cd $RELEASE_DIR/tmp/dom0_fs/etc/xen/
```

```
# =====================================================================
# Example bare metal Guest configuration
# =====================================================================
#
# This is a fairly minimal example of what is required for a
# Paravirtualised Linux guest. For a more complete guide see xl.cfg(5)

# Guest name
name = "bare1"

# Kernel image to boot
kernel = "/root/xzd_bare1.img"
```

```
# Initial memory allocation (MB)
memory = 128

# Number of VCPUS
vcpus = 1
```

6. Unmount and detach the dom0 filesystem.

```
$ cd $RELEASE_DIR
$ sudo umount /dev/loop0
$ sudo losetup -d /dev/loop0
```

7. Convert the dom0 FS RAW image back to a QCOW2 format

```
$ qemu-img convert -O qcow2 $RELEASE_DIR/XenZynqDist/images/dom0.img
$RELEASE_DIR/XenZynqDist/images/dom0.qcow
$ rm $RELEASE_DIR/XenZynqDist/images/dom0.img
```

7.2. Starting the new guest

Now that the bare metal guest has been added to the dom0 file system, it can be created as a guest. Follow Section 4.7 to boot the system. After that, start your bare metal guest using the following command:

1. Start your domain.

```
[root@xilinx-buildroot ~]# xl create /etc/xen/bare1.cfg -c
```

You should then see the following on the command line:

```
(d$u) XZD_Bare: Hello World
```

Where $u is the id number of the bare metal guest.

Chapter 8 Interacting with I/O Devices

8.1. Ethernet Pass-through

8.1.1. Introduction

Xen has the capability to pass direct memory addressable (DMA) devices via the system memory management unit (SMMU), giving guest domains direct access of the passed through hardware. This capability allows for increased speed, flexibility, and stability in the system. The instructions below provide a guide for how to configure the Xen system installed on the Xilinx UltraScale+ MPSoC (MPSoC). The instructions documented here are specific for the Ethernet device, however can be generalized to pass through any DMA-capable device via the SMMU.

It is assumed that the reader has an Ubuntu 12.04 host system and has downloaded the release image mentioned in chapter Chapter 3.

8.1.2. Modifying the Xen Device Tree

The first step in passing through an Ethernet device to a guest domain is to edit the xen.dts file located on the host computer in the XZD development system. The general process for enabling device pass-through is to disable the device from access to dom0 and to enable it for pass-through to a guest domain. Ethernet pass-through requires editing the two Ethernet devices in the xen.dts. We will disable the second Ethernet device and then enable it for device pass-through, but will also ensure that the Ethernet device assigned to domain 0 gets edited appropriately.

We need to convert the xen.dtb into a xen.dts file, which is a text file that we can edit. We will use the dtccommand on the host computer to create the dts file.

```
$ dtc -I dtb -O dts -o $RELEASE_DIR/xen.dts /tftpboot/xen.dtb
```

Open the xen.dts file and find the first and second Ethernet devices for the MPSoC in xen.dts. The found text in the xen.dts should look similar to that below.

```
    ethernet@ff0b0000 {
    #stream-id-cells = <0x1>;
    compatible = "cdns,gem";
    status = "okay";
    interrupt-parent = <0x1>;
    interrupts = <0x0 0x39 0x4 0x0 0x39 0x4>;
                  .
                  .
                  .
    ethernet@ff0c0000 {
    #stream-id-cells = <0x1>;
    compatible = "cdns,gem";
    interrupt-parent = <0x1>;
    interrupts = <0x0 0x4b 0x4>;
    reg = <0x0 0xff0c0000 0x1000>;
    clock-names = "pclk", "hclk", "tx_clk";
    clocks = <0x2 0x2 0x2>;
    #address-cells = <0x1>;
    #size-cells = <0x0>;
    status = "okay";
};
```

Figure 1: xen.dts

Once the Ethernet devices are located, then edit the xen.dts to look like the text below. In some cases, the tags *gem0* and *gem1* are already included, in other cases, it is not. Ensure that the tags, *gem0* and *gem1* are added to the file in the location below, paying attention to the highlighted portions of the file.

```
gem0: ethernet@ff0b0000 {
      #stream-id-cells = <0x1>;
      compatible = "cdns,gem";
      status = "okay";
      interrupt-parent = <0x1>;
      interrupts = <0x0 0x39 0x4 0x0 0x39 0x4>;
      reg = <0x0 0xff0b0000 0x1000>;
      clock-names = "pclk", "hclk", "tx_clk";
      clocks = <0x2 0x2 0x2>;
      #address-cells = <0x1>;
      #size-cells = <0x0>;
      local-mac-address = [00 0a 35 00 b2 02];
      phy-handle = <0x3>;
      phy-mode = "rgmii-id";

      phy@0 {
            reg = <0x0>;
            max-speed = <0x64>;
            linux,phandle = <0x3>;
            phandle = <0x3>;
      };
};

gem1: ethernet@ff0c0000 {
    #stream-id-cells = <0x1>;
    compatible = "cdns,gem";
    interrupt-parent = <0x1>;
    interrupts = <0x0 0x4b 0x4>;
    reg = <0x0 0xff0c0000 0x1000>;
    clock-names = "pclk", "hclk", "tx_clk";
    clocks = <0x2 0x2 0x2>;
    #address-cells = <0x1>;
    #size-cells = <0x0>;
    xen,passthrough = <1>;
    status = "disabled";
};
```

Figure 2: Edited xen.dts

Since the Ethernet is a DMA-capable device, we will add this to the SMMU by modifying the SMMU section of the xen.dts appropriately. Search and find the SMMU section and modify the code in the file to look like the section below.

```
            smmu0: smmu0@0xFD800000 {
                     compatible = "arm,mmu-500";
                     reg = <0x0 0xfd800000 0x20000>;
                     #global-interrupts = <1>;
                     interrupt-parent = <&gic>;
                     interrupts = <0 157 4>,
                             <0 157 4>, <0 157 4>, <0 157 4>, <0 157 4>,
                             <0 157 4>, <0 157 4>, <0 157 4>, <0 157 4>,
                             <0 157 4>, <0 157 4>, <0 157 4>, <0 157 4>,
                             <0 157 4>, <0 157 4>, <0 157 4>, <0 157 4>;
                      mmu-masters = <&gem0 0x874 &gem1 0x875>;
             };
```

Figure 3: SMMU Modifications

After the dts has been modified, we need to compile the dts back into the dtb and ensure that it is the saved in the correct location. Enter the following command to perform this task.

```
$ dtc -I dts -O dtb -o /tftpboot/xen.dtb $RELEASE_DIR/xen.dts
```

8.1.3. Modifying the Domain Configuration File

The next two files that require modification are done on the dom0 file system from the XZD development system. Locate the configuration file for the guest domain that will be using the passed-through Ethernet device. You will need to refer to section **Error! Reference source not found.** to convert the qcow image into a dom0.img. You will also need to follow the steps in chapter 4.3, section **Error! Reference source not found.** to mount the converted file system image. This will provide a file system that will contain the files for you to modify in the next few sections. Assuming you followed the instructions in section 4.3, edit your dom1.cfg, located in $RELEASE_DIR/tmp/dom0_fs/fs/dom0/etc/xen, by adding the text below to the bottom of the file. In addition to adding the text below, make sure that you either comment out or delete the virtual device at approximately line 39. This line begins with 'vif'.

```
# Ethernet Device
# This will set up the Ethernet so that it is
# accessible to this guest domain.
dtdev = [ "/amba/ethernet@ff0c0000" ]
device_tree = "/etc/xen/xen-partial.dtb"
irqs = [ 91 ]
iomem = [ "0xff0c0,1" ]
```

Figure 4: Editing the guest domain configuration file

The lines above add gem1 exclusively to the guest domain, in this case, domain 1. The options needed for pass-through are defined below:

dtdev - The absolute path of the device to pass-through in the device tree
device_tree - dom0 path to partial device tree to be passed to the guest
irqs - irq to be given to the guest
iomem = the physical pages to be passed in to the guest

8.1.4. Creating a Partial Device Tree

The last step in setting up the system for pass-through involves creating a device tree for the domain called a *partial device tree*.

Create a new file and name it *xen-partial.dts*. Ensure that this file is located in the **device_tree** path indicated in the configuration file shown in figure 4. The entire *xen-partial.dts* should be saved in $RELEASE_DIR/tmp/dom0_fs/fs/dom0/etc/xen and look like the one below.

```
/dts-v1/;

/ {
    #address-cells = <0x2>;
    #size-cells = <0x1>;

    passthrough {
        compatible = "simple-bus";
        ranges;
        #address-cells = <0x2>;
        #size-cells = <0x1>;

        misc_clk {
            #clock-cells = <0x0>;
            clock-frequency = <0x17d7840>;
            compatible = "fixed-clock";
            linux,phandle = <0x2>;
            phandle = <0x2>;
        };

        ethernet@ff0c0000 {
            #stream-id-cells = <0x1>;
            compatible = "cdns,gem";
            reg = <0x0 0xff0c0000 0x1000>;
            interrupts = <0x0 0x3b 0x4 0x0 0x3b 0x4>;
            clock-names = "pclk", "hclk", "tx_clk";
            clocks = <0x2 0x2 0x2>;
            #address-cells = <0x1>;
            #size-cells = <0x0>;
            local-mac-address = [00 0a 35 00 00 01];
            phy-handle = <0x1>;
            phy-mode = "rgmii-id";

            phy@0 {
                reg = <0x0>;
                max-speed = <0x64>;
                linux,phandle = <0x1>;
                phandle = <0x1>;
            };
        };
    };
};
```

Figure 5: xen-partial.dts

The *xen-partial.dts* now needs to be compiled into a binary file known as a device tree blob (dtb). We will use the host's dtc compiler, mentioned above, to compile the dts file into a dtb file.

Generate the dtb file by entering the following command:

```
$ dtc -I dts -O dtb -o xen-partial.dtb xen-partial.dts
```

Figure 6: Compile the DTS

Once all the files have been edited, you need to convert the dom0.img back into the qcow image. This can be done by following the steps in section 4.6.

8.1.5. Communicating with the Domains

You are now ready to boot both domain 0 and domain 1, each one is assigned a unique Ethernet device. To test the Ethernet device pass-through in the QEMU environment, one can ping a domain when logged into another domain and visa-versa.

One can also use the *nc,* or *netcat,* command that tests the ability to communicate between the domains and the host computer. To log into the host computer from a domain, enter the following command on your host computer.

```
$ nc -l 4321
```

This will create a quick server that can be used to communicate to other systems via an IP address. On one of the domains, enter the following command.

```
# telnet 10.0.2.2 4321
```

This will set up a connection with the host and allow you to send characters between the domain and the host computer. Entering ctrl-c in the domain will break the connection. Now follow the same steps in the other domain to test the other Ethernet device.

9.1. Section To Be Written

10.1. Current Support Options

For more information and support, please visit our web site.

http://dornerworks.com/services/xilinxxen

Details on the Xilinx Zynq UltraScale+ MPSoC can be found here:

http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html